Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/IL2005/000247

International filing date: 02 March 2005 (02.03.2005)

Document type: Certified copy of priority document

Document details: Country/Office: US
Number: 60/548,855
Filing date: 02 March 2004 (02.03.2004)

Date of receipt at the International Bureau: 07 March 2006 (07.03.2006)

Remark: Priority document submitted or transmitted to the International Bureau but not in compliance with Rule 17.1(a) or (b)

PCT/IL 2005 / 000247
21 FEB 2006

PA 1424492

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

February 09, 2006

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

APPLICATION NUMBER: *60/548,855*
FILING DATE: *March 02, 2004*

THE COUNTRY CODE AND NUMBER OF YOUR PRIORITY APPLICATION, TO BE USED FOR FILING ABROAD UNDER THE PARIS CONVENTION, IS *US60/548,855*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

[illegible]
T. WALLACE
Certifying Officer

23026 U.S.PTO

22154 U.S.PTO
60/548855
030204

PTO/SB/16 (01-04)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

Express Mail Label No.

INVENTOR(S)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
~~[illegible]~~ Hayim	Shaul	Tel-Aviv Israel

Additional inventors are being named on the ________ separately numbered sheets attached hereto

TITLE OF THE INVENTION (500 characters max)

A method and a system for reducing bandwidth by packet caching and backward referencing

CORRESPONDENCE ADDRESS

Direct all correspondence to:

☐ Customer Number:

OR

☒ Firm or Individual Name	Hayim Shaul		
Address	Rama 22A		
Address			
City	Tel-Aviv	State	Zip 63186
Country	Israel	Telephone	~~Fax~~ +972-3-648-7699

ENCLOSED APPLICATION PARTS (check all that apply)

☒ Specification Number of Pages 10
☐ Drawing(s) Number of Sheets
☐ Application Data Sheet. See 37 CFR 1.76
☐ CD(s), Number
☐ Other (specify)

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT

☒ Applicant claims small entity status. See 37 CFR 1.27.
☐ A check or money order is enclosed to cover the filing fees.
☐ The Director is herby authorized to charge filing fees or credit any overpayment to Deposit Account Number:
☒ Payment by credit card. Form PTO-2038 is attached.

FILING FEE Amount ($): 80

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.
☐ Yes, the name of the U.S. Government agency and the Government contract number are:

[Page 1 of 2]

Respectfully submitted,

SIGNATURE Hayim

TYPED or PRINTED NAME HAYIM SHAUL

TELEPHONE +972-3-648-7699

Date Feb 19th 2004

REGISTRATION NO. (if appropriate)

Docket Number:

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

PROVISIONAL APPLICATION COVER SHEET
Additional Page

PTO/SB/16 (08-03)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Docket Number	

INVENTOR(S)/APPLICANT(S)		
Given Name (first and middle [if any])	Family or Surname	Residence (City and either State or Foreign Country)
Hagim	Shaol	Tel-Aviv Israel

[Page 2 of 2]

Number 1 of 1

A method and a system for reducing bandwidth by packet caching and backward referencing

1 Abstract

This invention relates to a method and a system for reducing the bandwidth of data transmitted on a communication line. The technology that lies in the core of this invention is not compression nor is it regular caching. Both compression and caching are not very suitable for reducing bandwidth in today's communication lines that transfer Internet content.

Reducing bandwidth is a major desire of ISP-s (Internet service provider), home-users, content providers and almost every organization that owns a network. When such an organization connects its local net to a distant one (for example the Internet), it leases a communication line from a bandwidth provider – a Telco. Such a provider rents his pre-laid infrastructure to the organization, where the price is determined by the amount of bandwidth the organization wishes to transfer. Given this constellation, the desire for lower bandwidth is obvious. Less bandwidth means less money spent on renting a communication line.

The objective of this invention is to provide a method and a system that is effective and easy to use. The invention exploits the fact that a large quantity sent on a line repeats itself. By keeping a glossary of transmitted data, it is able to avoid transmitting the same part more than once. Nevertheless, this invention does not use compression, or caching to do so.

2 Background

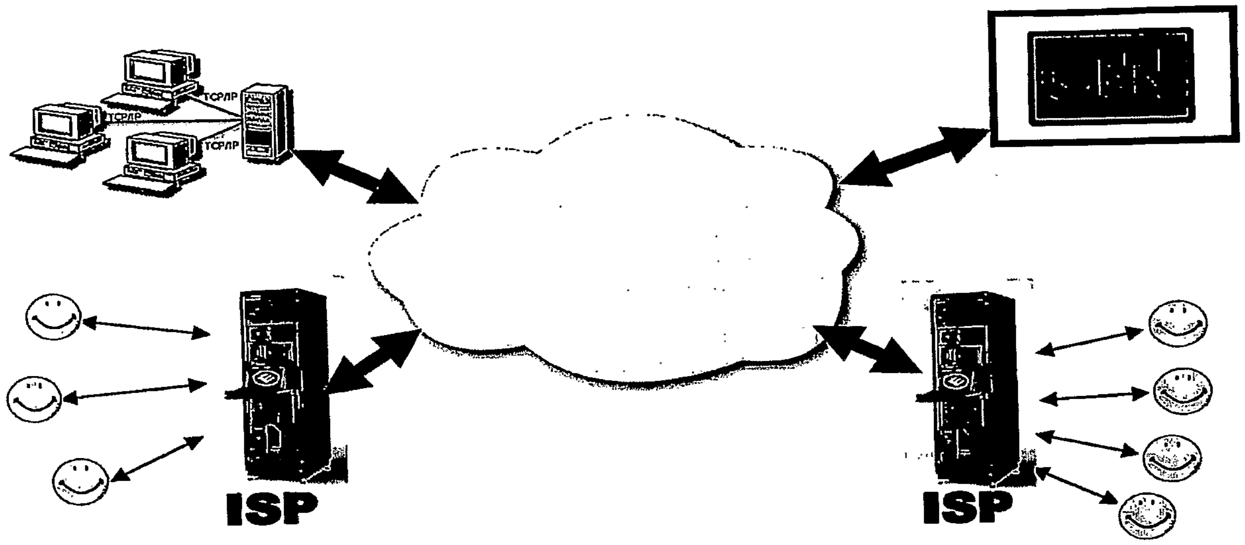
2.1 Internet

Computer communication is achieved by connecting two or more computers by a medium (such as a wire), defining a protocol of how data should be sent over the medium and how data should be received over the medium. Such a collection of computers is called a local area network (LAN).

The Internet today is a collection of LANs connected to each other by communication links. These links are usually long distance (across a continent, Trans Atlantic, etc.) and are very expensive to be lay. When an organization wishes to connect its LAN to a distant LAN it leases a long distance line that connects the two LANs.

An infrastructure of such long distance links is lay by bandwidth companies that lease the use of this infrastructure to organizations. The price for leasing the line is determined by the bandwidth the organization wishes to transfer, and by the distance of the line.

The following figure shows four networks (CNN's network, two ISPs and one network of an organization). These four networks are connected to the Internet backbone (represented by the cloud) with a communication line.



2.2 Internet Protocol – IP

A communication protocol is a set of rules that determine how communication is achieved over a line. The most common protocol for communication over the Internet is the Internet Protocol (also known as IP).

When sending a file or a stream using IP, this protocol partitions the file or stream into small blocks of data, called **packets**. These packets are usually no more than 1500 bytes long. An average mp3 file, which is three mega-bytes long, is partitioned into 2,000 packets.

IP address is an address given to every computer in the Internet. When sending a packet (usually as a part of a file or stream being sent) the IP address of the target computer needs to be stated in the prefix of the IP packet (also called **header**). The bandwidth provider delivers the packet to its destination, by using the target IP address stated in the packet header. The IP protocol states that the header should contain the IP addresses of the target computer as well as the IP address of the source computer.

Delivering a packet to a target computer is not enough. The packet needs to be delivered to a specific application run on the target computer. A logical address, called **port**, is defined for every application that intends to use communication. The target port needs to be stated in the prefix of the packet. It is written in the header of a protocol encapsulated by the IP, usually TCP or UDP. The TCP/IP and UDP/IP protocols state that the TCP or UDP header should contain the port number of the target application and the port number of the source application.

The four-tuple of source IP address, source port number, target IP address and target port number is an identifier to the communication. There are no two communications at the same tome with the same four values. In fact, upon receiving a packet the target computer determines the communication by these four values.

Links provided by bandwidth suppliers are not reliable. Packets sent might be **dropped** and never reached their destination, or they might come in a different order than sent. It follows then that the IP protocol is not reliable, as it provides no means for reliability. The protocols encapsulated by IP provide reliability. Different protocols provide different levels of reliability, by applying different techniques. Mainly these techniques include **retransmissions** of packets that are suspected to be lost, and sending acknowledges (**ack**-s) for packets that have arrived.

Other communication protocols (such as IPX, AppleTalk etc.) although different follow the same general outline.

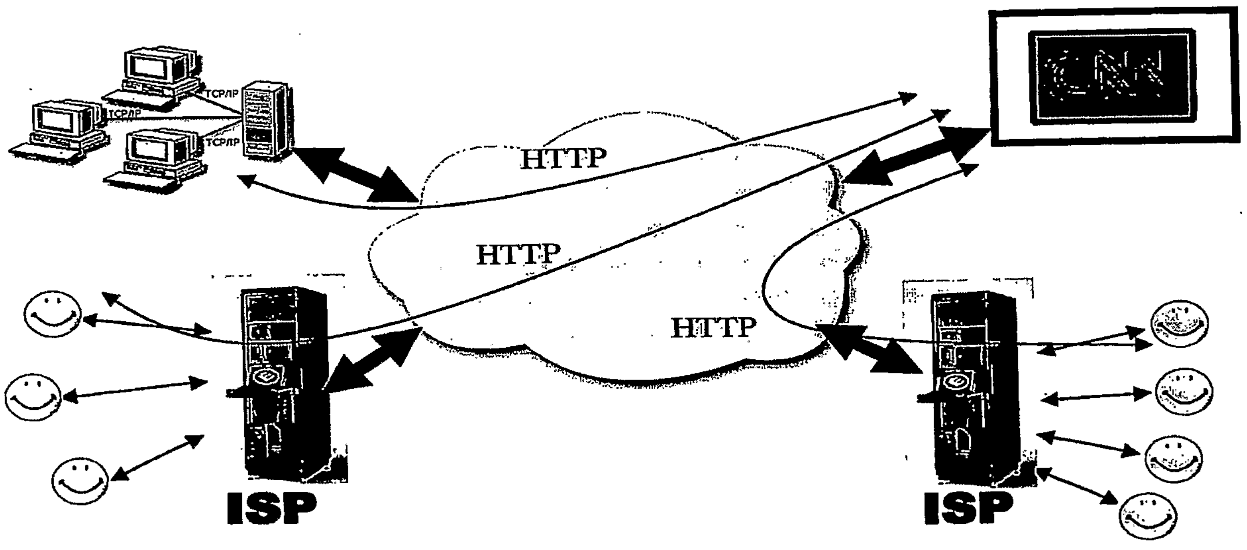
2.3 HTTP

HTTP is a file transfer protocol. It is used when browsing the Internet. HTTP gives every file in the world a unique name called **URL**. This name consists of the name of the computer storing the file, the directory path where the file is located and the name of the file.

The HTTP protocol states that a computer (which we call the client) sends an HTTP request for a file. The client states the URL of the requested file. The request, then, travels to the computer keeping the file (which we call server). The server's name is known since it is a part of the URL. The server then locates the file on its local disk and sends it to the client.

HTTP is the main protocol used for browsing the Internet. When a browser accesses a page on the Internet, it sends an HTTP request for that page. For example, when a browser accesses CNN's home page it sends an HTTP request for the URL www.cnn.com/default.html. Here www.cnn.com is the server's name and default.html is the file's name (in the main directory).

The following figure shows three networks, where in every network there is a computer requesting a file from CNN with HTTP protocol.



2.4 Peer-to-Peer network

A Peer-to-Peer (also **p2p**) network is a group of two or more computers, where any two computers can communicate with each other. A Peer-to-Peer file-sharing network is a p2p network, that every member of the network allows other members of the network to search its disk (or part of it) and download files. There are many p2p file-sharing networks that are different in the way a member joins the network, or searches for a file, but the main ideas are the same.

When a computer joins a p2p file-sharing network, it discovers several members of the network and announces itself to them. The way these members are discovered is dependant of the p2p network. A common technique is to publish a list of members at some page in the Internet. After a computer joins the network (by announcing itself) it can search for files and download them (as well as being search by others, and being downloaded from). A search is done by issuing a search request to the network members it is connected to. This search request is propagated by computers who get it until a certain depth. A list of files that satisfy the search term is then returned to the computer that issued the search. This list contains a list of file names, as well as the computers that currently keep them. Upon receiving the list, the computer starts a communication directly with a computer holding the file. Most p2p file-sharing networks allow a file to be downloaded from several computers simultaneously, by partitioning the file into several fragments and downloading each fragment from a different computer.

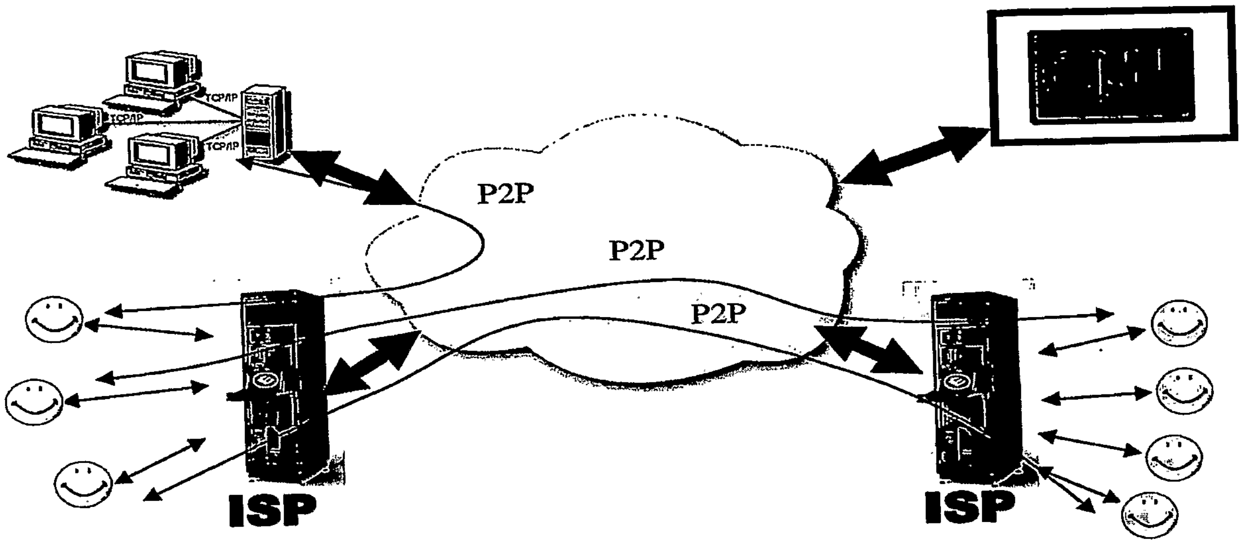
There is a big difference between p2p file-sharing and browsing. P2P file sharing, as oppose to browsing downloads the same file every time from a different computer, whereas, when browsing the same file is downloaded from the same computer. In fact, the computer to download from is given by the URL.

The following figure shows the previous network, but now, it shows three p2p connections.

2.5 Caching

One method of reducing bandwidth is caching. The idea behind caching is to store files that were downloaded recently, for the chance that they will be downloaded again. A cache stores the recently downloaded files indexed by their URLs. When a cache intercepts a request for a file, it answers it from its local storage (if possible), instead of accessing the remote server.

The advantages of caches are clear. A request is served locally without having to download the file again from the server. The disadvantages of caches are also obvious. The cache will not find the file stored, if it changed its name (as is the case in p2p file-sharing networks, where the file is downloaded from a different server every time). Caches might also provide a wrong version of the file, if its content has changed.



2.6 Compression

Another method for reducing bandwidth is compression. When transmitting a file (or a stream), upon transmitting a block, compression makes use of previous data sent to anticipate the continuation of the data. For example in the English language, the letter 'q' is always followed by the letter 'u'. After transmitting the letter 'q', in an English text, we need to transmit the next letter only if is different than 'u'. Compression techniques learn the data as they transmit it and develop a model for it.

There are two disadvantages with compression. The first is that most of the Internet content is already compressed, and compressed data cannot be compressed again. Compressed data has no model that can anticipate it, if there had been such a model, the compressor that created the compressed file would have used it. The second disadvantage is the computation power needed to create a model of the data being transmitted is very high. As a result compression techniques cannot deal very well with large amounts of data, or high-speed communications.

3 The Method

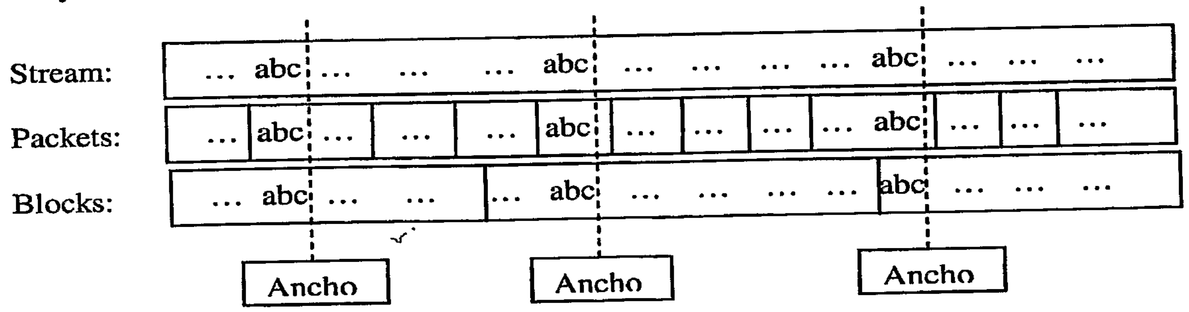
The invention consists of a method for reducing bandwidth by packet caching, and a system for reducing bandwidth by packet caching. The core of the invention is of storing packets and retrieving them fast in an efficient way. We now describe the method for storing packets and then we describe how a packet can be retrieved efficiently.

Throughout the following description we regard a file being transferred as a stream of data. This is a reasonable consideration since devices in the net do not know in advance the file that is being transferred and they learn the file as it is passed through them, just as a stream.

A device applying the method learns the method as it reads packets belonging to the stream from the net. We partition a stream that we learn into **data blocks**. The size of a block is independent of the packet size. Hereafter we consider a block to be 64K (although it can be any size whatsoever). A block also need not start at a beginning of packet. As we read packets of a stream from the net we copy their data into a data block. When a block is full, and contains 64K of data we write it to the disk after performing some preprocessing we mention below. Another way to determine the ending position of a block and the beginning of the next block in a stream is by using anchors (which will be explained later). Using anchors we can ensure that blocks will always be partitioned in the same way.

We define a **hash key,** to be a number of *n* bits that depends on the value, that its size is *m* bytes, where the probability of having two identical hash keys for two different *m*-byte values is very low. Hash keys can be created by computing the CRC value of *m* bytes, or by calculating their SHA1 value, DES value, or any other function known to satisfy the above condition. The decision of the values of *n* and *m* depends on the network and the packet type that the method is applied on. We use hash keys in two functions, for locating a block on the disk and for locating a packet in the block. For finding a block on the disk we chose a 64-bit hash taken on 100 bytes and for finding a packet in the block we use a 16-bit hash taken of 5 bytes.

We define an **anchor** to be a position in the stream that is dependent only of small amount of data (few bytes), and not dependent of the starting position of the block that contains the anchor. Nor is the anchor dependent on the starting position of the packet that contains the anchor. An example for defining anchors on a stream is choosing an anchor to be every position in the stream where the string "abc" appears. See figure below. Another example for defining anchors is choosing anchors to be every position in the stream where a 9-bit hash of 5 consecutive bytes is zero. We chose to use a 9-bit CRC because given a CRC of five byte string it is easy to remove the contribution of the first byte in the string and add a new byte at the end of the string. Thus we "roll" the CRC over the buffer efficiently. In order to prevent too many block ids we skip four hundred bytes after finding an anchor.



Stream:
... abc ... ... abc ... ... abc ... ...
Packets:
... abc ... ... abc ... ... abc ... ...
Blocks:
... abc ... ... abc ... ... abc ... ...
Ancho
Ancho
Ancho

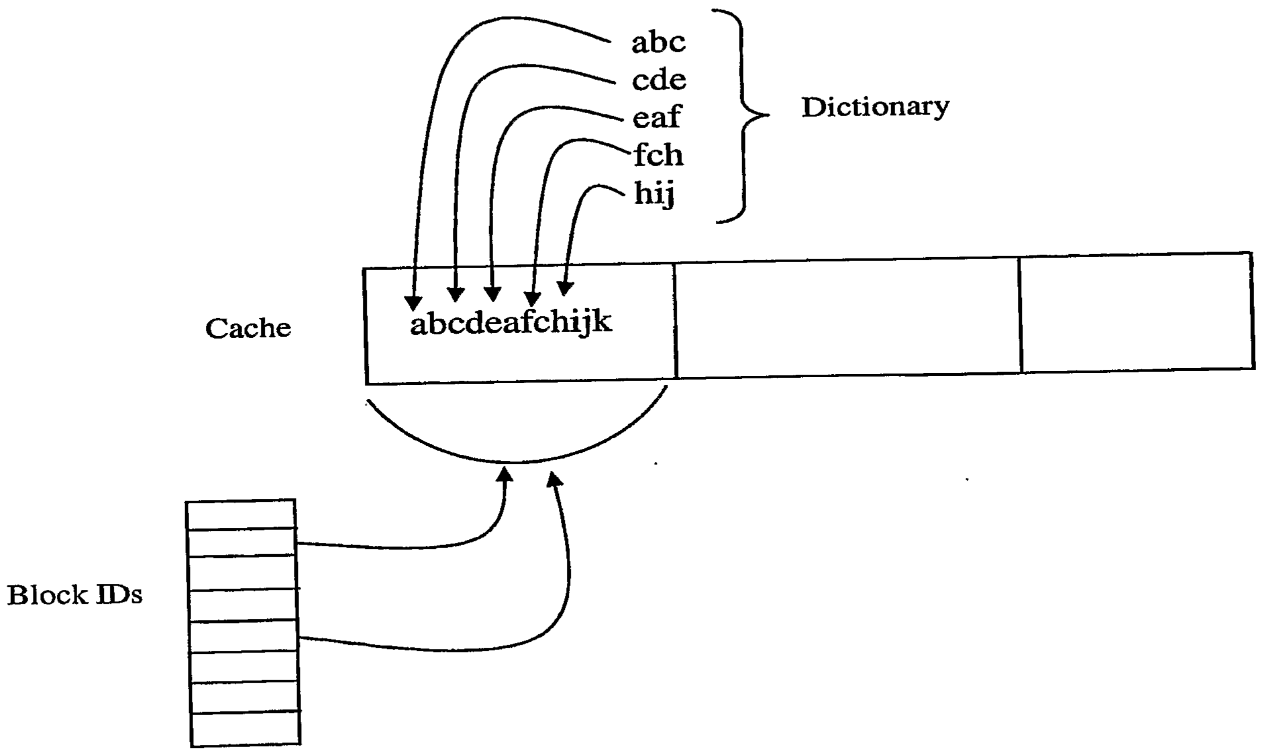
Every place we find an anchor, we compute a 64-bit hash over the next 100 consecutive bytes. We call this value **block ID**. Obviously a block has many IDs. We make sure that a block does not have too many IDs by ignoring anchors that are less than 500 bytes far from the previous anchor we considered. In is clear that a packet holds no more than three hash ids, and a block holds no more than 128 IDs. Because of the properties of the way we choose anchors, we expect three anchors in every packet, and 128 anchors in every block. Thus, three block ids in every packet and 128 block ids in every block.

We keep all the block ids in an array on the disk. We call this array the **hash array**. We define a function that maps every block id to one entry of the hash array (although many ids might be mapped to the same entry). At each entry we keep a list of block ids together with the location of the locations of their blocks.

For every block, we compute hash keys for every *m* consecutive bytes in the block and store every hash key together with the position where it was generated in an array. We call this array a **dictionary** of the block. As mentioned earlier the hash key in this key is 16 bits long, and it is calculated over 5 consecutive bytes. We store these values in an array, but they can be stored in a list, a tree or any structure that allows efficient searching. We set the dictionary to be an array of 65536 entries. If we calculate a hash key *h* at position *p*, we set the *h*-th entry of the array to hold the number *p*. To find the position where a hash key *h* was computed we look the value stored in the *h*-th entry in the array.

To reduce the dictionary size we don't compute a hash key for every *m* consecutive bytes, but only for those whose starting position inside the packet can be divided by *x*, where *x* is a parameter that can be chosen by the developer. A higher value of *x* will require a smaller dictionary size. We chose *x* to be 16.

The following figure shows all the structures we introduced, and their relations.



This figure shows three blocks stored in the cache. The first block contains the string "abcdeafchijk". We also show the dictionary of the first block. The dictionary shows where triplets of character appear in the block. The figure shows also an array of block IDs, two of which belong to the first block. Upon receiving a packet, we search the packet for an anchor, after finding an anchor we compute the block ID by computing a hash value of the 100 bytes following the anchor. We use the block ID we got to look at an array storing block IDs. This array also stores the location in the cache in which the block is stored. After fetching the block into memory, we use the dictionary to find large substrings of the packet in the block. Then we delete the substrings from the packet and replace them with references to the block.

We now show how to retrieve a packet from the cache, and how to use this method to reduce bandwidth. We keep two computers at each end of a communication line. For simplicity of explanation we assume that all communication are transmitted from the same end of the line, and received at the other end. We also assume that the two computers at the two ends have run for a sufficient amount of time, have studied the information transmitted over the line, and have built the data structures explained above.

A packet we read from the network is a part of a stream of communication. This stream of communication is distinguished from other communications by its communication id, which is the four tuple: source IP address, destination IP address, source port and destination port. Upon reading the packet before it is transmitted, our device goes over the packet to find anchors in this packet. From the way we constructed the anchors, it can be shown that the expected number of anchors in a packet is three. This means that there is a very high probability that some anchor will be found. Notice that the position of the anchor in the stream does not depend on its position in the packet. This guarantees that the anchor we found in the packet is the same anchor that was found when we learned the stream.

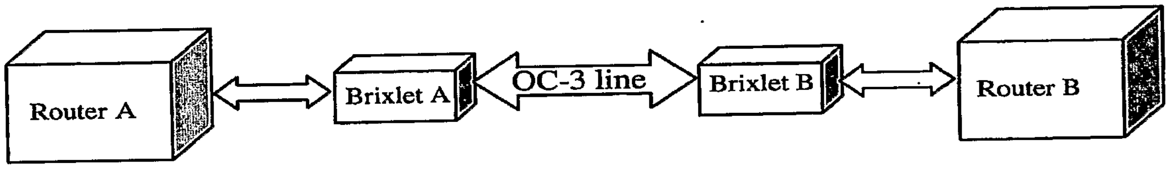
After finding one such anchor we compute the block id that is defined at that anchor. We use the hash array to find the position of block on the disk. More specifically, we read the entry in the array that stores the block id we computed. Since there are many block ids that are mapped to this entry, it contains a list of block IDs together with the positions of the blocks on the disk. We search the list for the bock ID we computed. If we found the block ID in the list we fetch the block from the disk. We transmit the packet over the line, and we also send the device on the other end a message that tells it to fetch the same block from its disk. It takes the disk a few milliseconds to fetch the block. During this time, more packets of the same stream may be transmitted unchanged over the line. The number of these packets is not high (less than a dozen).

After the block has been fetched from the disk, and when a packet arrives from the same stream, we locate the position of the packet inside the stream using the dictionary. We compute a hash key h on five bytes inside the packet. We read the h-th entry of the dictionary. This entry holds the position where a string that generated the same hash has appeared in the block. We compare the data in the packet and the data in the block to see if they match. If they do, we replace the data in the packet with an indication that the data appears in the block together with its position in the block and its length. We repeat this procedure as many times as needed until we go over the entire packet. The computer at the receiving end of the line reconstructs the packet by copying data from the block into the packet.

To improve the fetching time of blocks, we apply prefetching techniques. We try to prefetch a block before it is actually needed. We do this by identifying that the stream reached the end of the current block, and then prefetch a set of blocks that we predict that one of them will be needed next. For this, we need to study for every block a list of blocks that are needed after it is used.

4 The system

The system comprises of two computers, one computer at every end of a communication line. The communication that is transmitted on the line passes through both of the computers. The two computers study the files and streams that are transmitted over the line. They partition them into blocks and store the blocks on the disk together with a dictionary. They also update the hash file. When a packet of a stream is transferred, the two computers search their disk, using their hash file, and fetch a block that was stored previously. This block is used by the transmitting computer to replace data in the packet with references to data inside the block, and by the receiving computer to reconstruct the packet according to the references.



5 Conclusions

We have shown how to cache blocks of files and streams, where the developer can chose the size of the blocks. This method and system can cache fractions of files and stream to be used in the future. This method can be used to reduce communication traffic of peer-to-peer applications. Normal caches do not work for p2p traffic since they need a URL, a unique name given to every file, whereas this method is not dependent of the file's name (if such one exists). This method caches the contents of the stream based on hash value that depends only on the contents of the stream.

This method is unaware of the streams it caches. It is unaware if they are p2p traffic, http connections, or any other type of communications. Also, it is virtually impossible to reconstruct a file from the blocks stored at the cache, without having the complete file. This is especially true for files sent in p2p manner, since these files are partitioned to fragments and each fragment is downloaded from a different peer in a different communication.

Because this method caches blocks of files and streams, it can be used to reduce bandwidth of new files based on similar files that are stored in the cache. A file is partitioned into blocks, each of which is stored separately. When a different file that has many common substrings with the file stored (as is the case of html pages that were written with the same template), the file that was stored will be fetched and the new file will be compared to the old file, and a reduction will be obtained, even though it is the first time the new packet is transmitted.

What is claimed is:

1. A method for partitioning a stream into blocks, in such a way that if the stream is partitioned into packets in a different way, the blocks will be constructed in the same way.
2. A method for giving labels to blocks in a way that if the stream is partitioned into packets in a different way the same labels will be assigned to the same blocks.
3. The method of claim 1 where the blocks are partitioned with anchors.
4. The method of claim 1 where the blocks are partitioned by size.
5. The method of claim 2 where no more than a certain number of labels can be generated for one block.

6. A system for partitioning a stream into blocks, in such a way that if the stream is partitioned into packets in a different way, the blocks will be constructed in the same way.
7. A system for giving labels to blocks in a way that if the stream is partitioned into packets in a different way the same labels will be assigned to the same blocks.
8. The system of claim 6 where the blocks are partitioned with anchors.
9. The system of claim 6 where the blocks are partitioned by size.
10. The system of claim 7 where no more than a certain number of labels can be generated for one block.